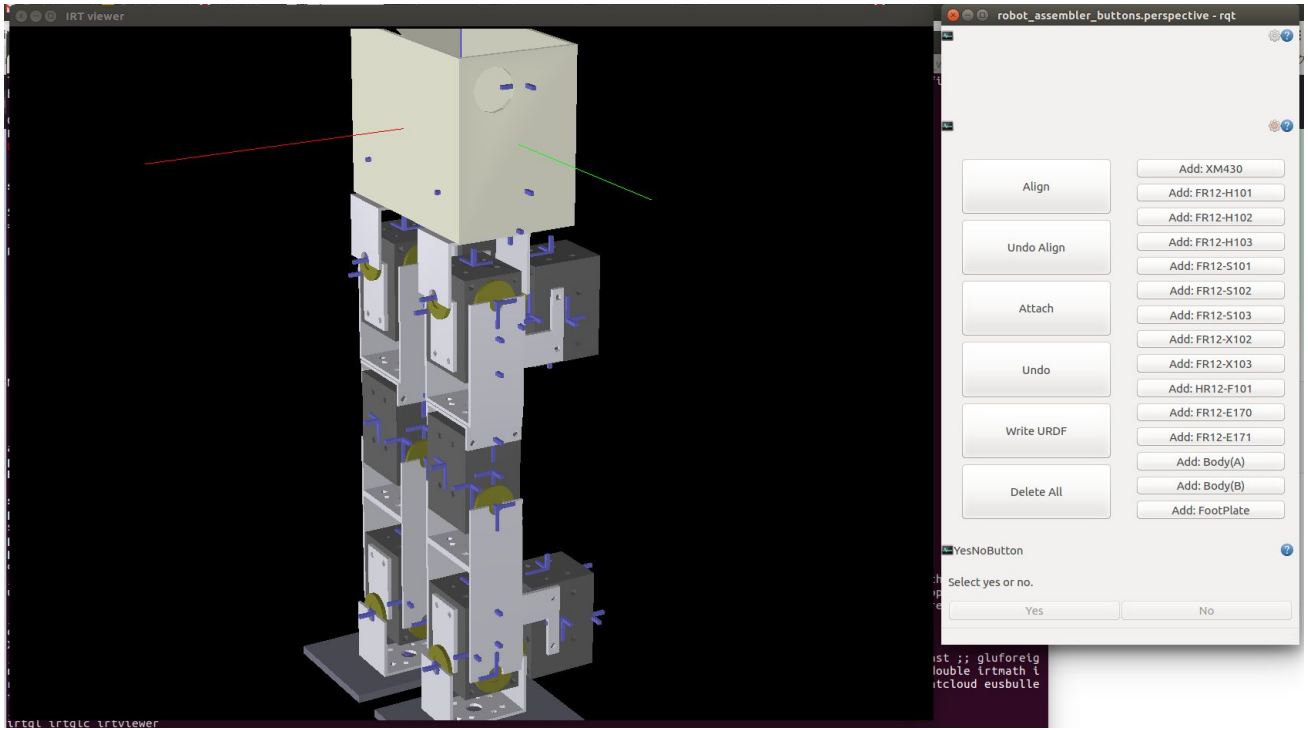
Robot Assembler説明

Robot Assembler

自作のロボットモデルを簡単に作るためのプログラム

GUIで部品を選択し、3D画面で配置を決めていけばモデルファイルを作ることができる



Install

robot_assembler

githubレポジトリ / https://github.com/agent-system/robot_assembler

ROSのインストールが済んでいる場合
rosinstall のファイル
https://github.com/agent-system/robot_assembler/raw/master/config/robot_assembler.rosinstall

mkdir new_ws; cd new_ws
$ wstool init src \
https://github.com/agent-system/robot_assembler/raw/master/config/robot_assembler.rosinstall
またはマージの場合
$ wstool merge -t src \
https://github.com/agent-system/robot_assembler/raw/master/config/robot_assembler.rosinstall

をして、
rosdep install -q -y -r --from-paths src --ignore-src

catkin build robot_assembler
source devel/setup.bash

Dockerの準備

Dockerのインストール

ここを参考にDockerをインストール

(Nvidiaのドライバを使っている人はnvidia-dockerのインストールもお願いします)

https://github.com/YoheiKakiuchi/robotsimulation-docker

Windows / MacOSの場合は（あまり詳細な質問に対応できないかもしれない）

- vmwareなどを使用してubuntuを使う
- Docker for Windows / MacOS を使う

なにか問題があったら、

https://github.com/agent-system/robot_assembler の issue に質問してください。

Dockerの準備

1. パターン1 最小限の構成

docker pull yoheikakiuchi/robot_assembler

wget https://raw.githubusercontent.com/agent-system/robot_assembler/master/scripts/run_docker.sh

wget https://raw.githubusercontent.com/agent-system/robot_assembler/master/scripts/exec_docker.sh

2. パターン2

git clone https://github.com/agent-system/robot_assembler.git

をしてきて

robot_assembler/scripts ディレクトリで

docker build -f Dockerfile -t yoheikakiuchi/robot_assembler .

とbuildする

Robot Assembler の起動

Ubuntuの環境を使っていてインストールができたら
  roslaunch robot_assembler robot_assembler.launch

Dockerの環境なら
  ./run_docker.sh

Windows等の環境なら
  docker run --net host -it yoheikakiuchi/robot_assembler bash
 docker内で
  source /catkin_ws/devel/setup.bash
  DISPLAY=xx.xx.xx.xx:0 ## xserverのIP
  roslaunch robot_assembler robot_assembler.launch

Xwindow サーバーが必要になります。
VcXsrv https://sourceforge.net/projects/vcxsrv/ など
xwindowのアクセスコントロールに気をつけて下さい。
http://k-hiura.cocolog-nifty.com/blog/2017/04/x-windowwin-lin.html

Robot Assembler の起動

dockerで上手く動かない場合

以下がミニマムなセットなのでこれで実行してみて下さい

```
 docker run --net host -it yoheikakiuchi/robot_assembler bash
docker内で
 source /catkin_ws/devel/setup.bash
 DISPLAY=xx.xx.xx.xx:0 ## xserverのIP(windowsのみ)
 roslaunch robot_assembler robot_assembler.launch
```

Robot Assembler画面

起動に成功すると

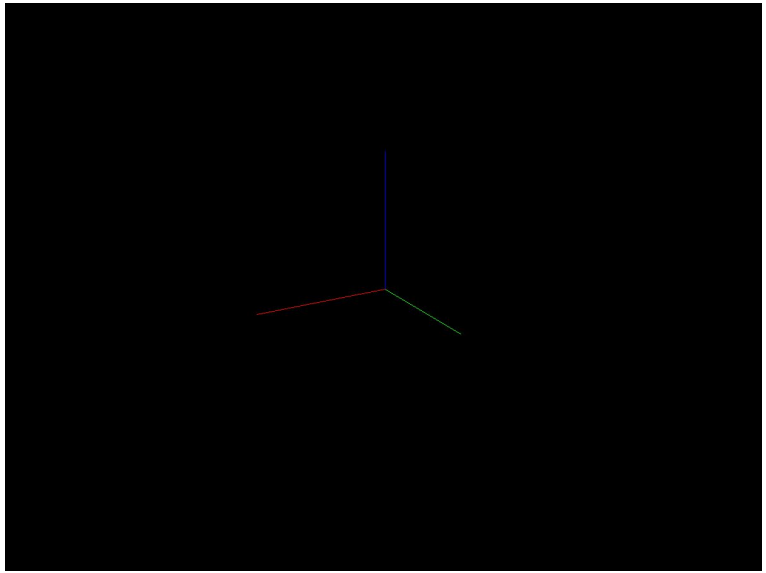
Eusの画面と、ボタンが現れる

(初期画面は真っ黒)

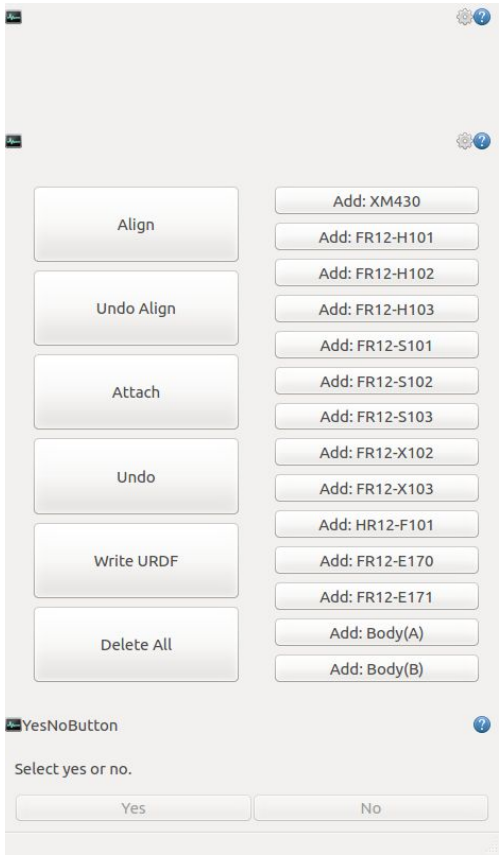
左クリック系はirtviewerそのまま

右クリックでrobot_assemblerの操作をする

3D画面



ボタンGUI

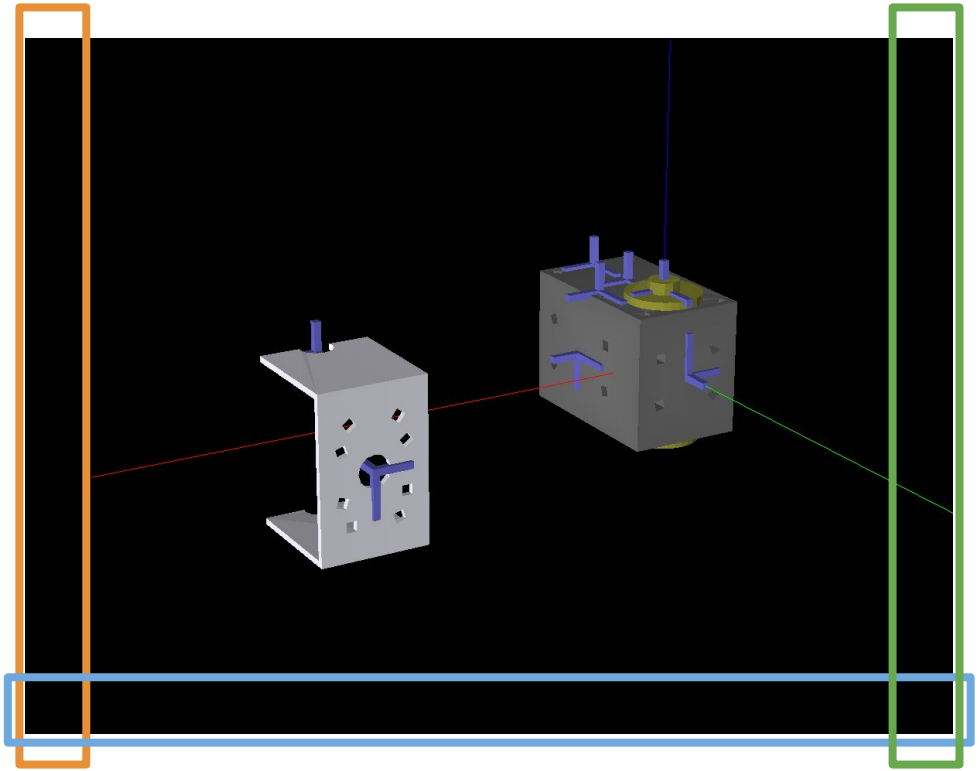


3D画面

左クリック系はirtviewer

- 左クリックしてドラッグすると視点が回転する
- 端をドラッグすると平行移動する
  左端は上下移動(オレンジ枠)
  下端は左右移動(青枠)
- 右端をドラッグするとズームする
  緑枠

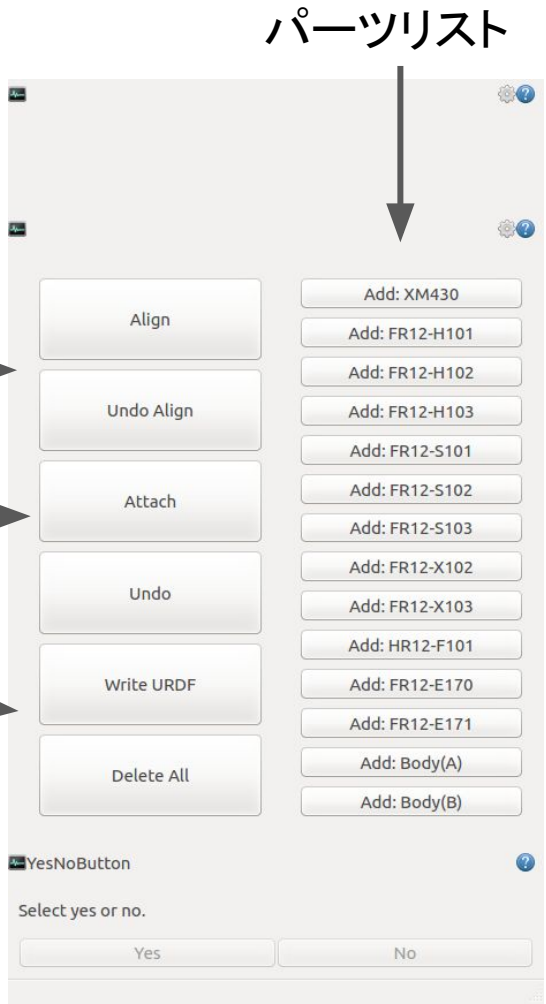
右クリックでrobot_assemblerの操作をする
矢印をクリックして指定する



ボタンGUI

ボタンの左側の列は操作になる（ボタンGUIは通常の左クリック）
--
Align: AttachせずにAttachした時の姿勢に並べる
　　複数回クリックすると固定可能な姿勢を toggleする
Undo Align: Alignを戻す
--
Attach: ロボットにパーツを固定する（一番最初はワールドに固定）
Undo: Attachを一回戻す
--
Write URDF: URDFファイルを書き出す
Delete All: 全部消して最初から始める

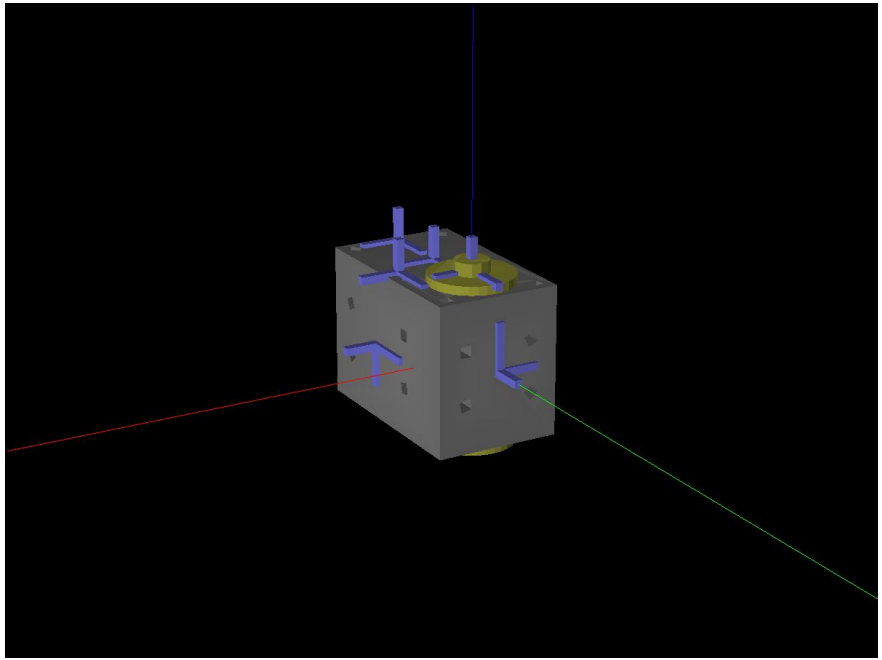
ボタンの右側の列は
パーツのリストで、押すとパーツが3D画面に現れる

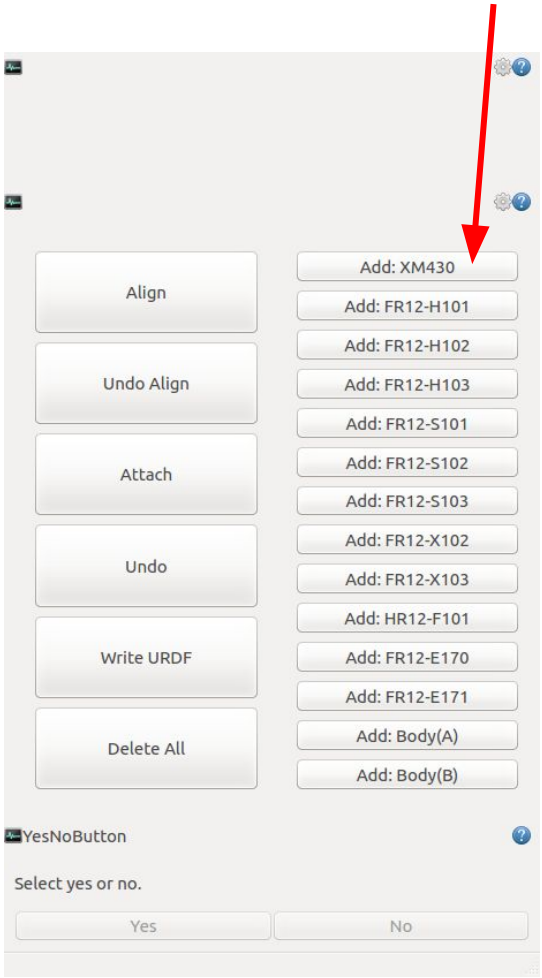


操作手順

Add:XM430
を押す

パーツが現れる

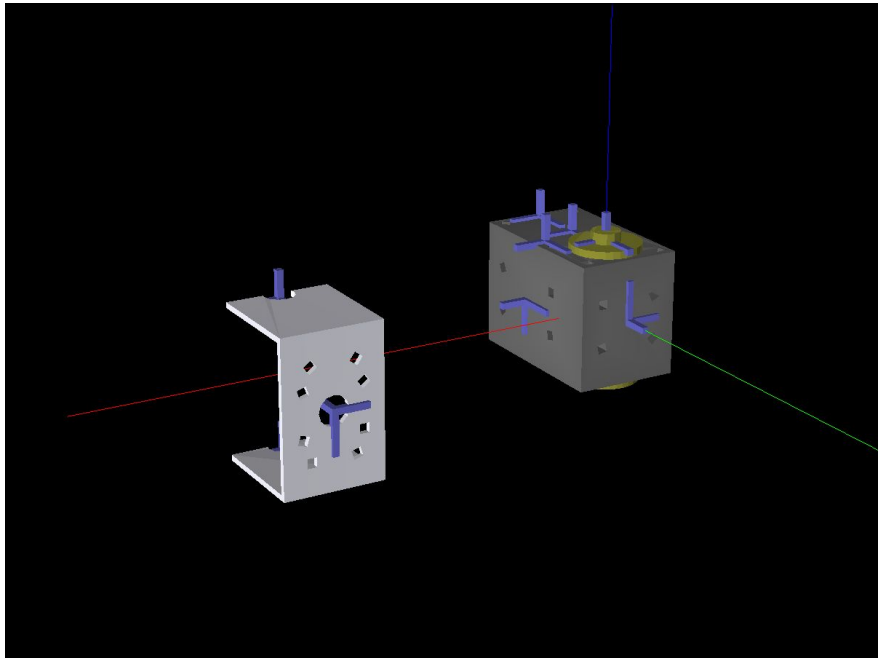


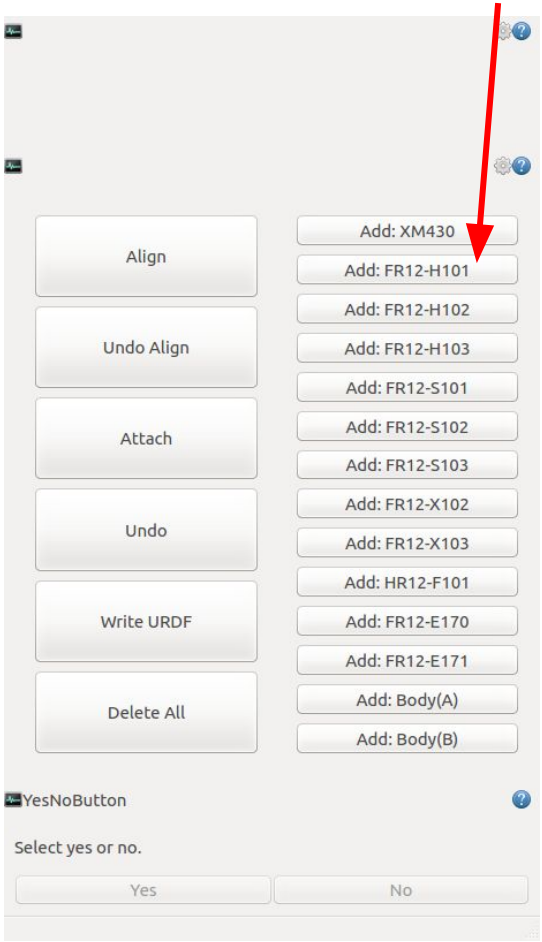


操作手順

Add:FR12-H101を押す

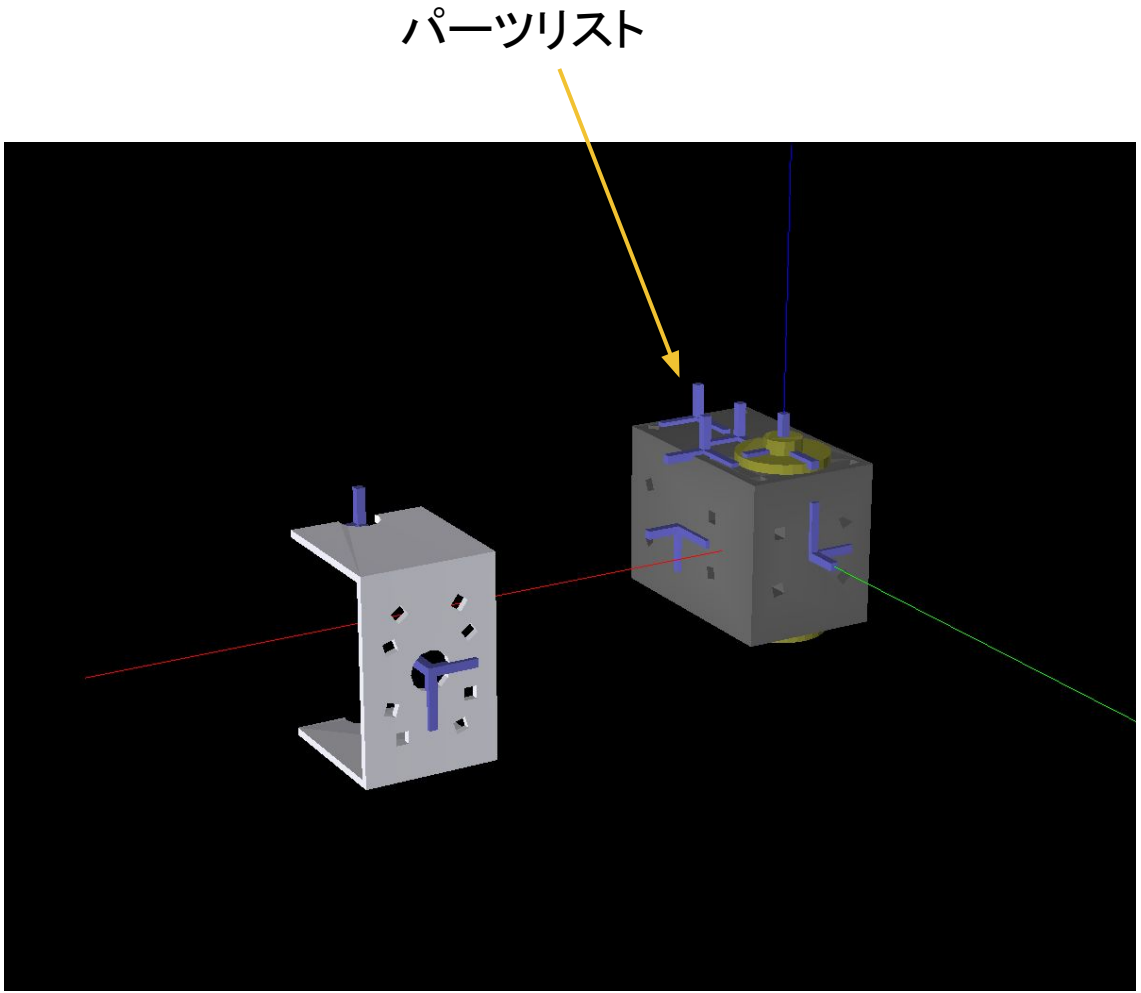
パーツが現れる





操作手順

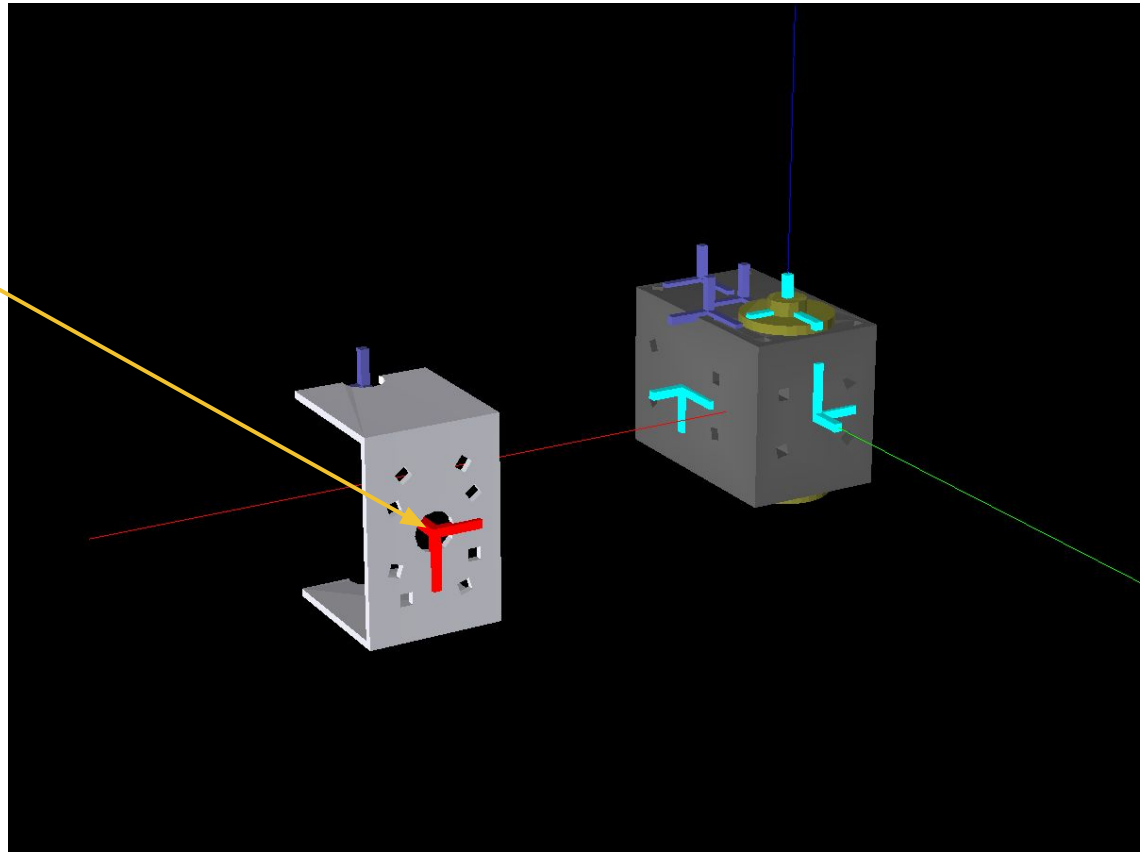
Add: xm430
と
Add: FR12-H101
を押すとパーツが現れる
紫の矢印は接続可能点



操作手順

ヒンジの紫矢印を選択する
(マウスの右クリック)

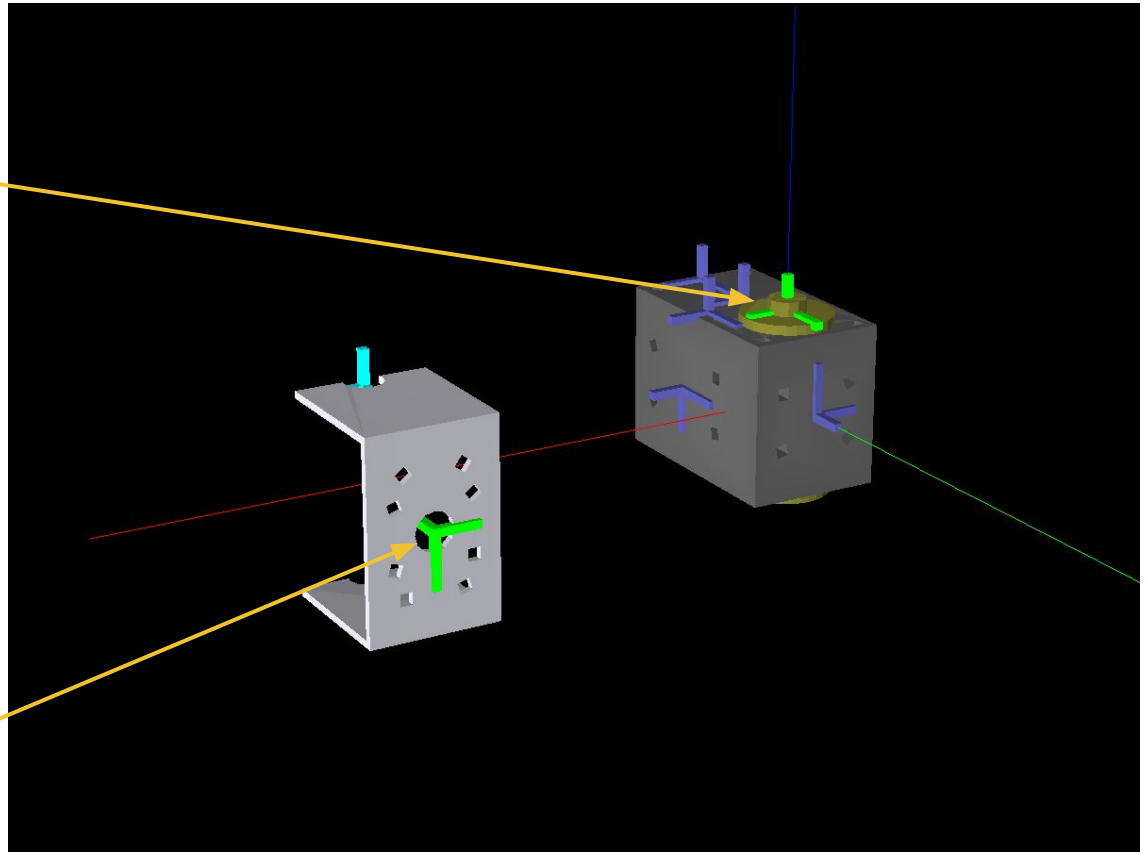
赤矢印になる
(対応点が選択されていない)
青空色の矢印は、赤矢印と対応可能な点



操作手順

ホーンをセレクトする
(マウスの右クリック)

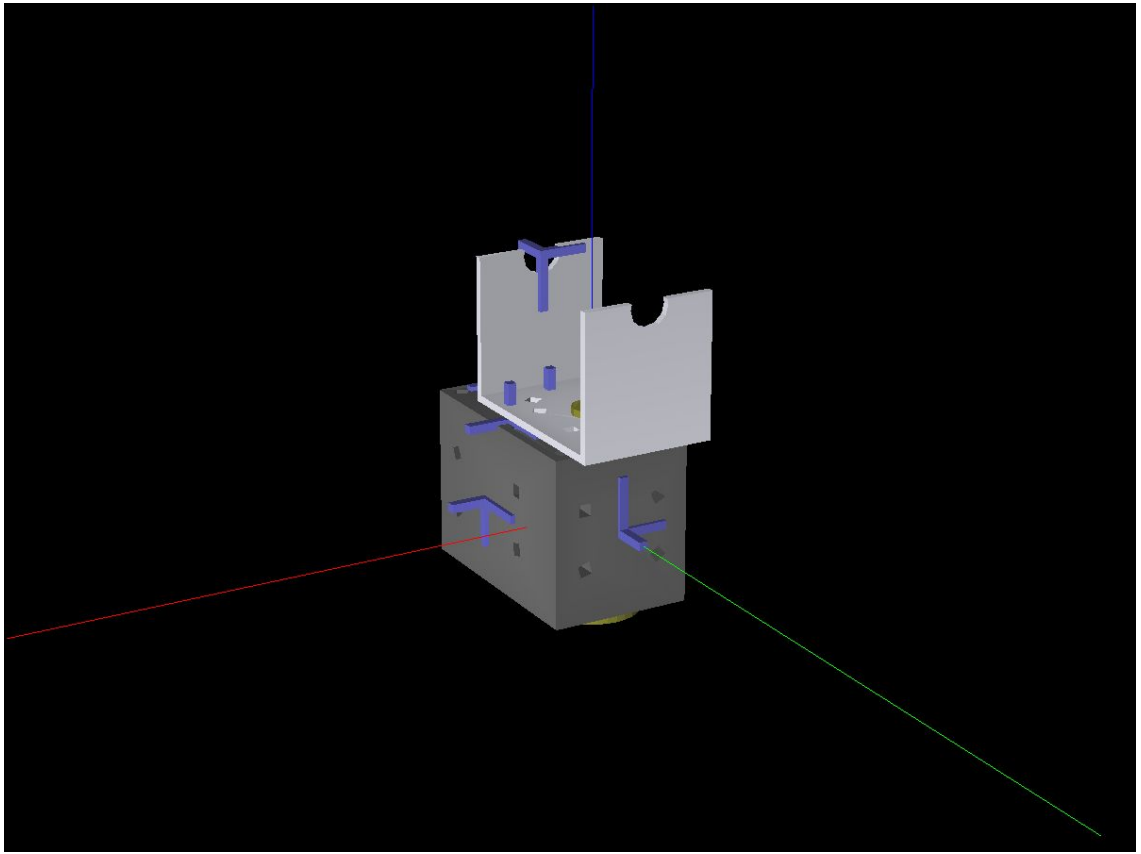
選択矢印が緑になる
この状態がAttach可能な状態



操作手順

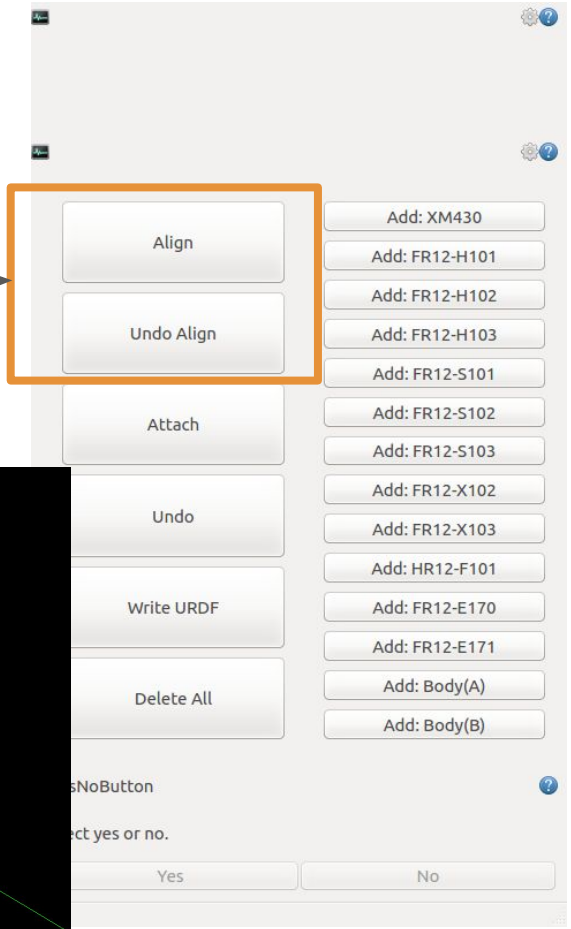
ボタンGUIのAttachを押すと
アタッチされる

ボタンGUIのAddでパーツを
追加して次を続ける

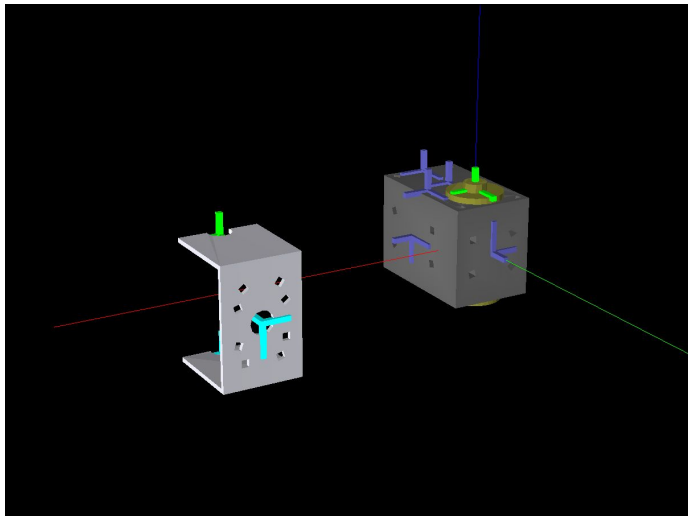


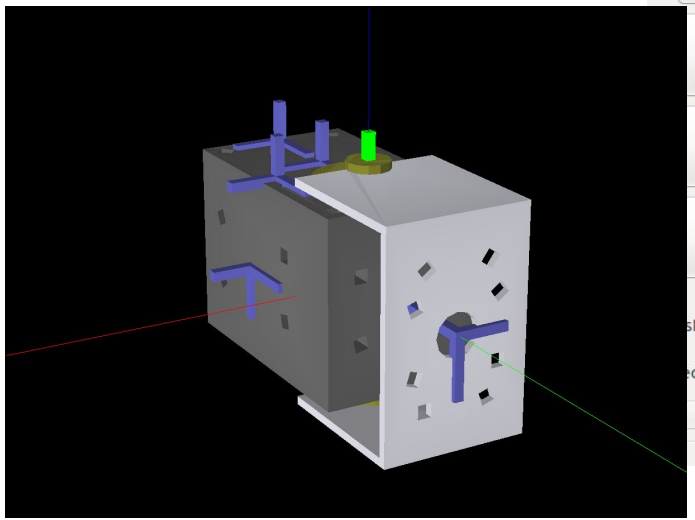
ボタンGUI (Align)

Align: AttachせずにAttachした時の姿勢に並べる
　　複数回クリックすると固定可能な姿勢を toggleする
Undo Align: Alignを戻す



Align

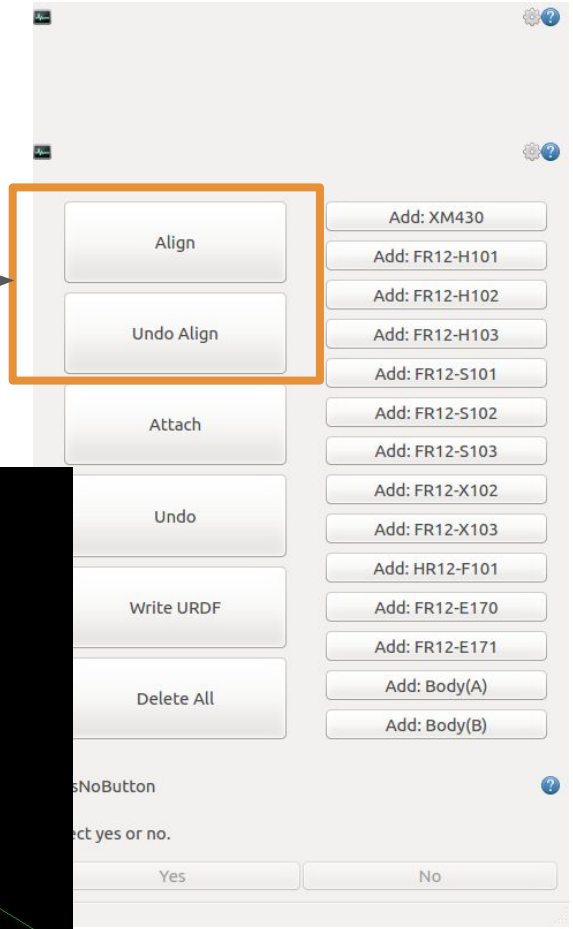




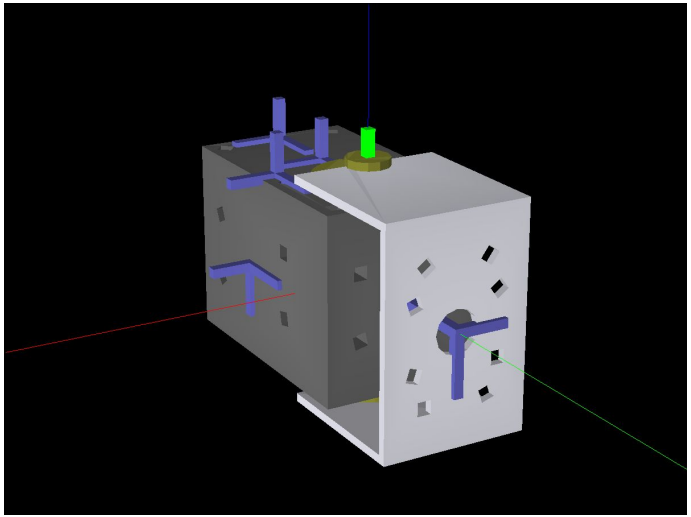
Undo align

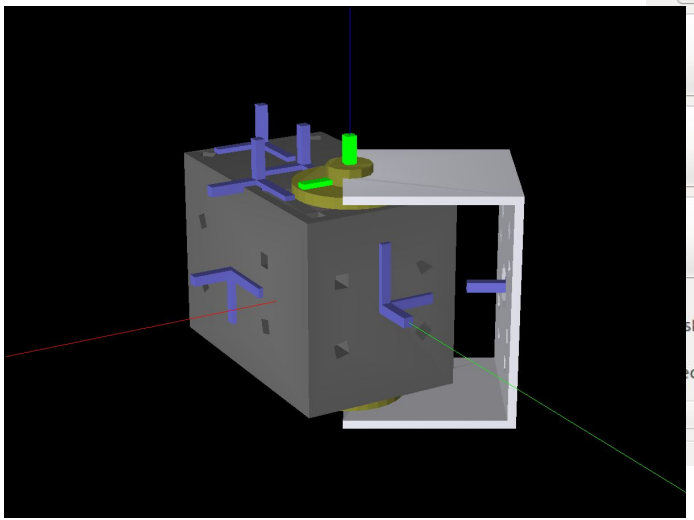
ボタンGUI (Align)

Align: AttachせずにAttachした時の姿勢に並べる
　　複数回クリックすると固定可能な姿勢を toggleする
Undo Align: Alignを戻す



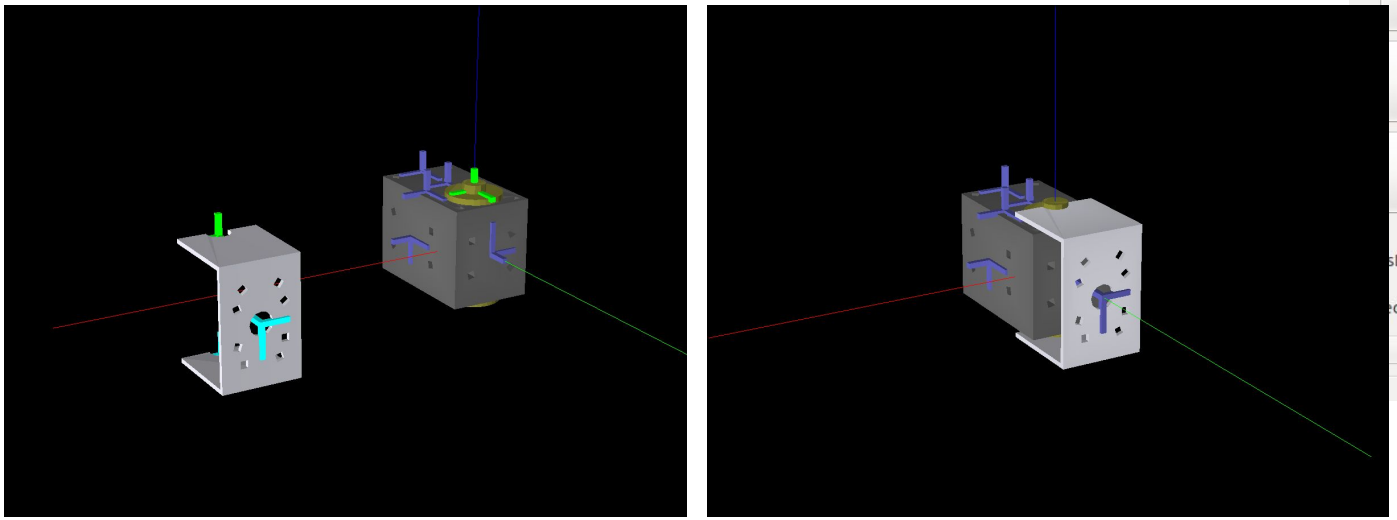
Alignを複数回押す

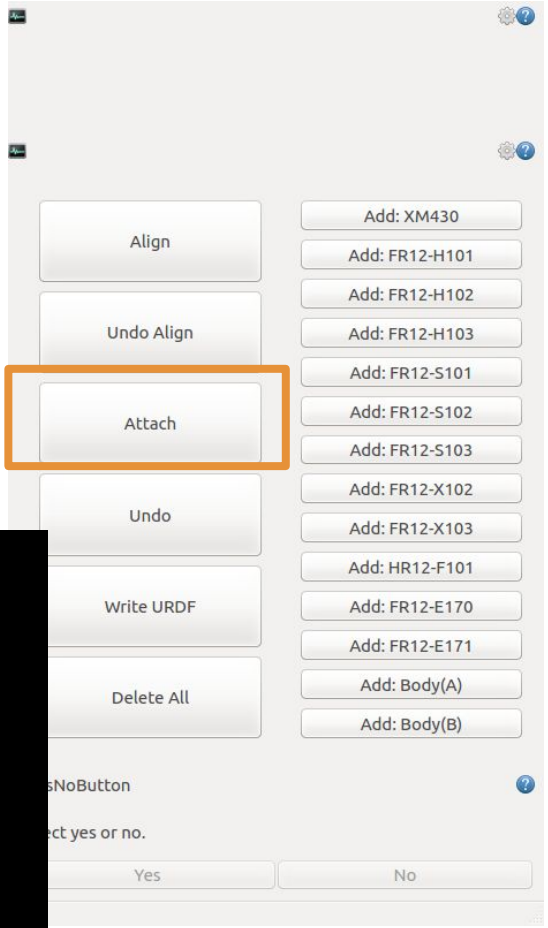




ボタンGUI (Attach)

Attach: ロボット(最初に追加したパーツ側)にパーツを固定する
　　（一番最初はワールドに固定）
Alignの後にも可能
(Alignは固定していないので、
　パーツボタンを押すとAlignしたパーツが 新しいパーツに置き換わる

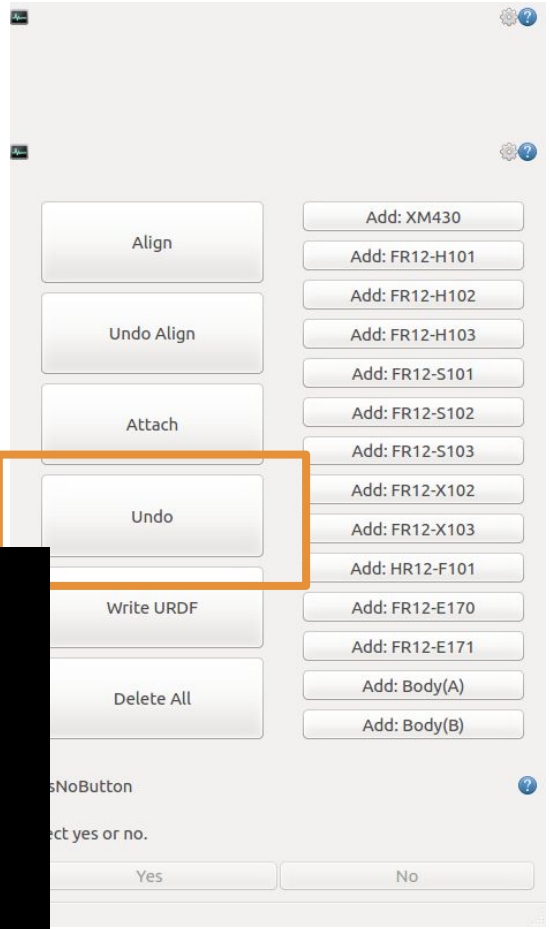


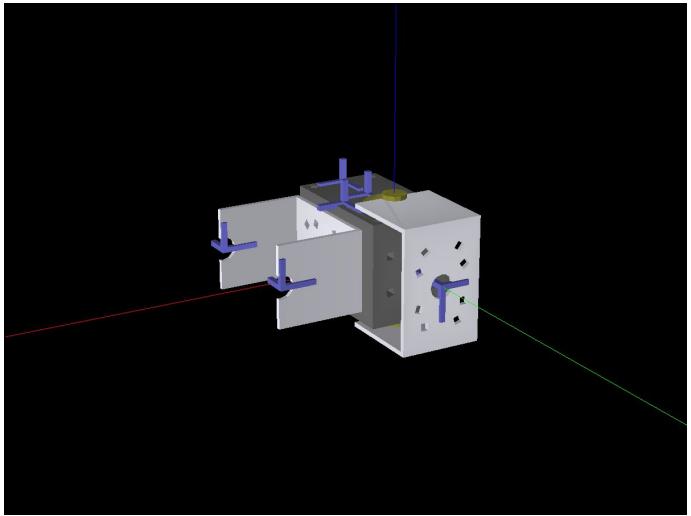


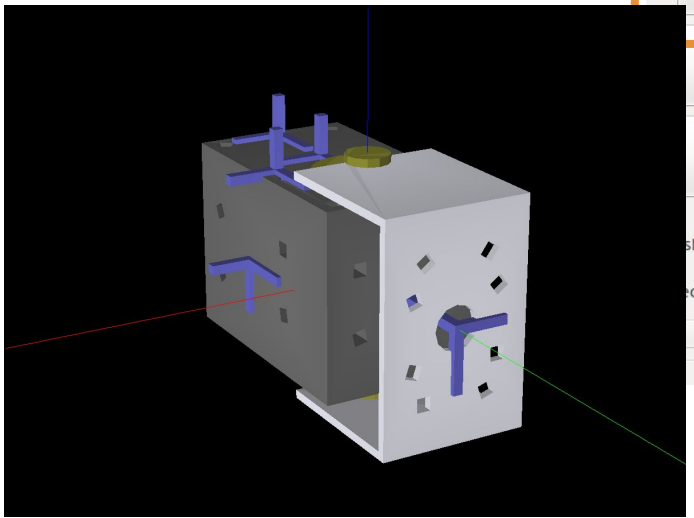
ボタンGUI (Attach)

Undo: Attachがひとつ戻る
複数回クリックすると順番に戻ってゆく

間違えたらUndo する







ボタンGUI (Write URDF)

Write URDF: URDFファイルを書き出す
Delete All: 全部消して最初から始める

Write URDF: すると
OUTPUT_DIR/の下に(defaultは /tmp dockerでは/userdir)
ROBOT_NAME.urdf (defaultのROBOT_NAMEはassembled_robot)
ROBOT_NAME.urdf.gz_controller.yaml
ROBOT_NAME.urdf.euscollada.yaml
ROBOT_NAME.roboasm.l

以下のようにして、ROBOT_NAMEとOUTPUT_DIRを変更できる
roslaunch robot_assembler robot_assembler.launch \
ROBOT_NAME:=your_robot_name OUTPUT_DIR:=path_to_you_want

以下のようにして、前回の生成物の続きから始めることができる
roslaunch robot_assembler robot_assembler.launch \
START_WITH:=ROBOT_NAME.roboasm.l

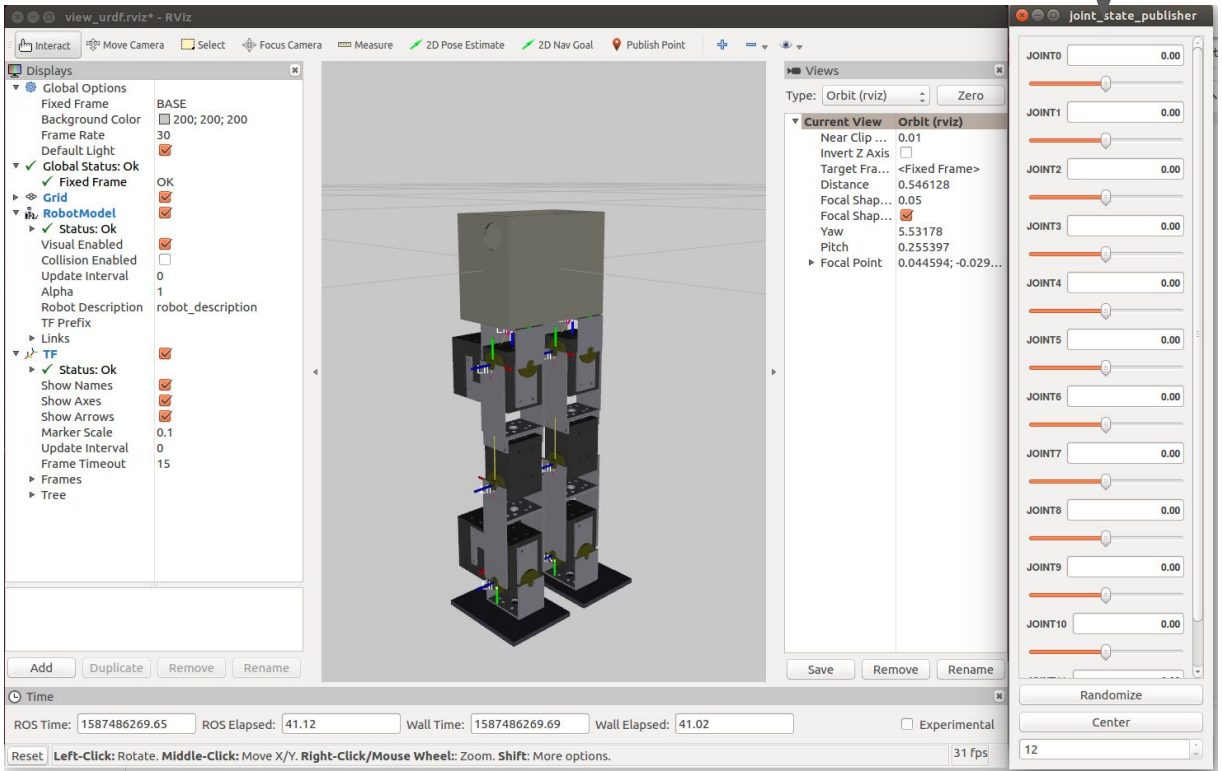
(ファイルやディレクトリは絶対パスの必要がある。
/userdir/ROBOT_NAME.roboasm.l や カレントディレクトリにあるなら $(pwd)/ROBOT_NAME.urdf)

Rviz

URDFが生成されたので、以下を実行すると rviz にモデルが出せる
roslaunch robot_assembler urdf_check.launch \
model:=/tmp/ROBOT_NAME_robot.urdf

関節角度を指定するGUIが出るので、
関節の名前と回転方向を確認する

関節角度 の指定



EusLispのロボットモデルを作る

OUTPUT_DIRにて、以下のコマンドでROBOT_NAME.l が生成される

rosrun euscollada collada2eus -I ROBOT_NAME.urdf -O ROBOT_NAME.l \

-C ROBOT_NAME.urdf.euscollada.yaml

roseus ROBOT_NAME.l

roseus$ (setq *robot* (ROBOT_NAME))

roseus$ (objects (list *robot*)) ;; ロボットが表示される

roseus$ (setq av (send *robot* :angle-vector)) ;; 関節角度のベクトルが得られる

roseus$ (setq (elt av 0) 60) ;; ベクトルの0番目の要素を60 [deg] に設定

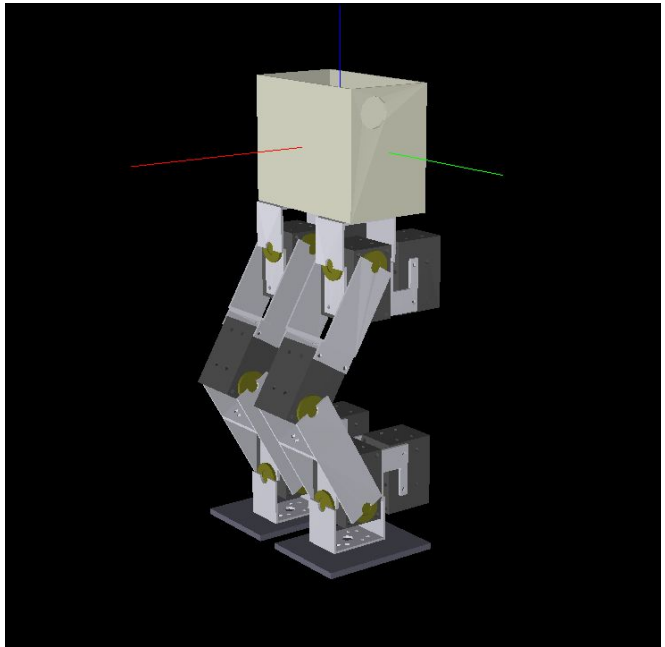
roseus$ (send *robot* :angle-vector av) ;; ロボットのモデルに反映

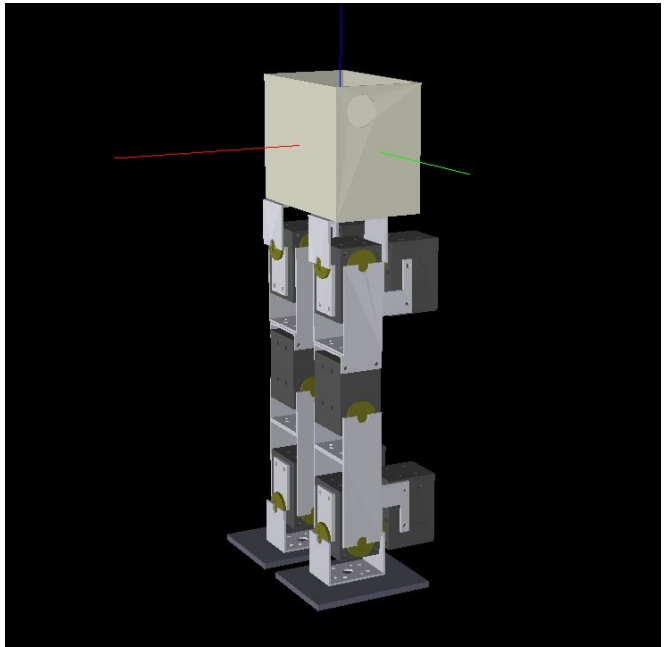
EusLispのロボットモデルを作る

sampleの例

robot_assemblerで組み立てたものと同じ（内部の表現は違う）モデルが表示される

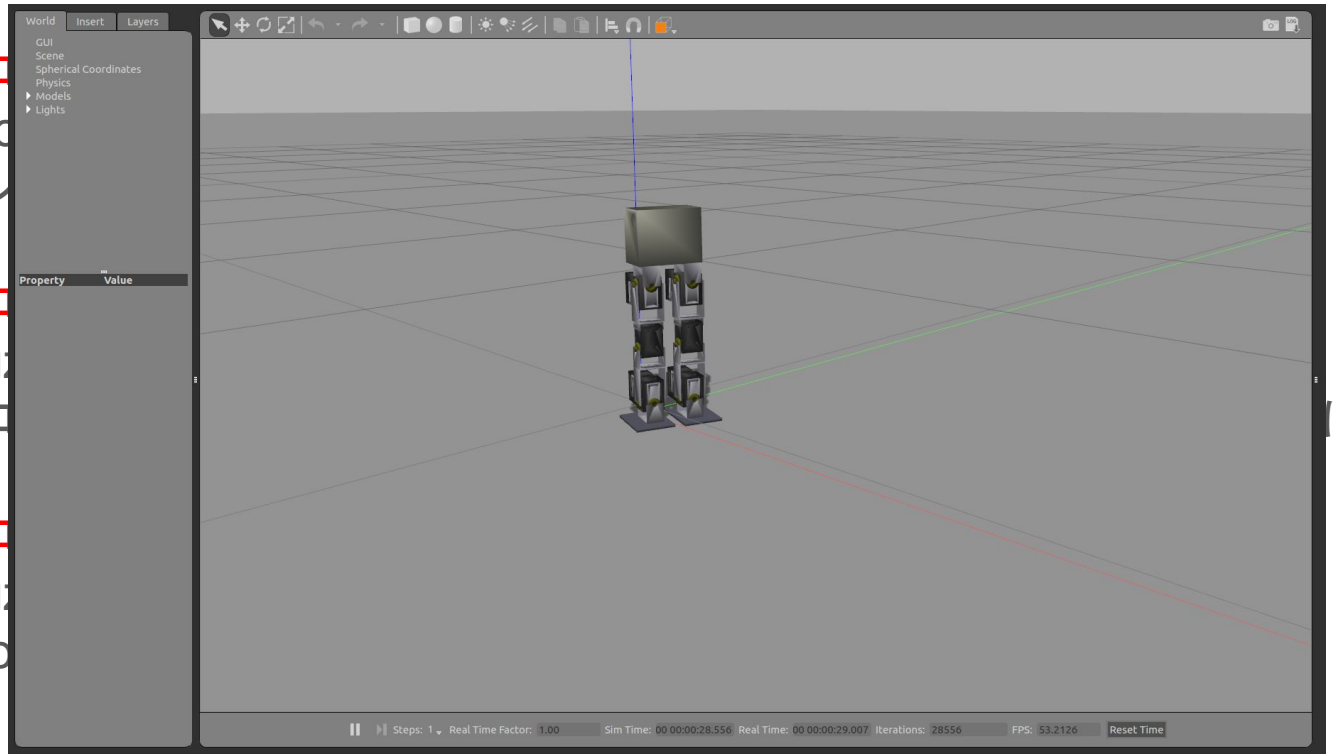
関節角度を変えることができる





Gazebo

roslaunch robot_assembler robot_assembler_gazebo.launch \
model:=/tmp/ROBOT_NAME_robot.urdf



は0.6)

Gazebo

roslaunch robot_assembler robot_assembler_gazebo.launch \
model:=/tmp/ROBOT_NAME_robot.urdf

ロボットが表示されない時は
robot_assembler_gazebo.launch の引数に paused:=true を追加
シミュレーターが停止して始まるのでロボットを探してみる

ロボットが地面にめり込んで/空中に浮いて始まるときは
gzpose:=”-z 1.0” を追加
ロボットがz方向に移動して始まるので 1.0を変えてみる (デフォルトは0.6)

ロボットの初期姿勢を変えたい時は
gzangles:="-J JOINT0 -0.2 -J JOINT1 0.2” を追加
joint名はrvizのところで確認

Gazebo内のロボットの操作方法

gazebo内のロボットを動かす(コマンドラインから)

rostopic pub /fullbody_controller/command trajectory_msgs/JointTrajectory "

joint_names: ['JOINT0', 'JOINT1', 'JOINT2', 'JOINT3'] ## すべてのjointを羅列してください

points:

- positions: [0.2, -0.2, 0.2, -0.2] ## position [rad] 関節の数と同じ要素数にしてください

  time_from_start: {secs: 3, nsecs: 0}" ## 3秒で目的の角度へ到達するように動く

yaml 形式のテキストです。

“{joint_names: ['JOINT0', 'JOINT1', 'JOINT2', 'JOINT3'], points: [{ positions: [0.2, -0.2, 0.2, -0.2], time_from_start: {secs: 3, nsecs: 0}}] }”

と同じになります。

positionsを増やすと、第１姿勢に移行してから、第２姿勢に移行するというようにできる

Gazebo内のロボットの操作方法

gazebo内のロボットを動かす(pythonインターフェース)

robot_assembler/gazebo/move_sample.py にサンプルがある

joint_namesを使うロボットの関節名に修正

```
traj_msg.trajectory.joint_names = ['JOINT0', 'JOINT1', 'JOINT2', 'JOINT3', 'JOINT4', 'JOINT5', 'JOINT6',
'JOINT7', 'JOINT8', 'JOINT9', 'JOINT10', 'JOINT11']
```

Trajectory.points の経由点を修正

```
traj_msg.trajectory.points.append(\
JointTrajectoryPoint(positions=[0, 0, 0.20, -0.40, 0.20, 0,   0, 0, 0.20, -0.40, 0.20, 0 ], #姿勢1
                                        time_from_start = rospy.Duration(2))) ## 前の姿勢から2sec
```

Gazebo内のロボットの操作方法

gazebo内のロボットを動かす(pythonインターフェース)

roseus

roseus$ (load "ROBOT_NAME.l");; 必ず下のファイルよりも先にロードしてください

roseus$ (load "package://robot_assembler/euslisp/assembled-robot-interface.l")

roseus$ (assembled-robot-init) ;; *robot* と *ri* ができます

roseus$ (objects (list *robot*)) ;; viewer にロボットが表示されます

;; gazebo内のロボットにviewer内のロボットと同じ姿勢になるように指令を送ります

roseus$ (send *ri* :angle-vector (send *robot* :angle-vector) 2000)

Sample

robot_assembler/sample にサンプルがある

Write URDFをしてURDFファイルを書き出し
roslaunch robot_assembler robot_assembler.launch ROBOT_NAME:=SAMPLE \
START_WITH:=$(rospack find robot_assembler)/sample/SAMPLE.roboasm.l

OUTPUT_DIRにURDFファイルができるので Rvizで見る
roslaunch robot_assembler urdf_check.launch \
model:=OUTPUT_DIR/SAMPLE.urdf

Gazeboを走らせてみる
roslaunch robot_assembler robot_assembler_gazebo.launch \
model:=OUTPUT_DIR/sample/SAMPLE.urdf

EusLispのモデルを作る
rosrun euscollada collada2eus -I SAMPLE.urdf -O SAMPLE.l -C SAMPLE.urdf.euscollada.yaml

Sample (dockerの場合)

robot_assembler/sample にサンプルがある

Write URDFをしてURDFファイルを書き出し
./run_docker.sh roslaunch robot_assembler robot_assembler.launch ROBOT_NAME:=SAMPLE \
OUTPUT_DIR:=/userdir START_WITH:=/catkin_ws/src/robot_assembler/sample/SAMPLE.roboasm.l

/userdir にURDFファイルができるので Rvizで見る
./run_docker.sh roslaunch robot_assembler urdf_check.launch model:=/userdir/SAMPLE.urdf

Gazeboを走らせてみる
./run_docker.sh roslaunch robot_assembler robot_assembler_gazebo.launch \
model:=/userdir/SAMPLE.urdf
Gazeboの動作は
./exec_docker.sh をしてdockerのターミナルを立ち上げてから試してください
EusLispのモデルを作る
./run_docker.sh rosrun euscollada collada2eus -I SAMPLE.urdf -O SAMPLE.l -C \
SAMPLE.urdf.euscollada.yaml

./run_docker.sh では カレントディレクトリが docker内の/userdirにマウントされます
docker内での生成物ができるディレクトリは /userdirを指定しましょう

robot_assembler ソースプログラム構成

robot_assembler
githubレポジトリ / https://github.com/agent-system/robot_assembler

config/robot_assembler_parts_settings.yaml ;; パーツ、接続可能点の設定
config/robot_assembler_buttons_layout.yaml ;; ボタンGUIの設定 (viewerとROSで通信)

euslisp/robot-assembler.l ;; 組み立てロボットのクラス定義等
euslisp/robot-assembler-viewer.l ;; viewerに関するプログラム
euslisp/robot-assembler-node.l ;; ROSのノード

launch/robot_assembler.launch ;; メインのランチャー

config/robot_assembler_parts_settings.yaml の説明

yaml記法のファイル (PyYamlで読めればよい)

```
;; 接続可能位置のtypeのリスト (A)
fixed-point-type-list: [horn12, horn12-hole,

;; 接続可能位置のtypeの組み合わせリスト (B)
fixed-point-type-match-list:
  -
    pair: [horn12, horn12-hole] ;; (A)で定義した名前の組み合わせを書く
    allowed-configuration: [fixed, ;; 回転可能などの定義のリスト、(C)で定義する

;; 接続時の回転の定義リスト (C)
pre-defined-configuration-list:
  -
    type: invert ;; (B)のリストのallowed-configurationで用いた名前
    description: 'rotate 180[deg] around z-axis'
    rotation: [0, 0, 1, 180] ;; どの方向に回転するか
```

config/robot_assembler_parts_settings.yaml の説明

```
;; アクチュエータがついた部品の定義
actuators:
  -
    type: xm430 ;; 形名
    geometry: ;; 形状
      -
        type: mesh ;; 今のところmeshのみ (assimpで読める種類)
        url: "meshes/xm430_dynamixel.dae"
        scale: 1000
    mass-param: ;; マスパラメータ (ここだけ m, kg系)
      mass: 0.1
      center-of-mass: [0.0, -0.012, 0.0015]
      inertia-tensor: [3.202708e-05, 0.0, 0.0,  0.0, 2.077708e-05, 0.0,  0.0, 0.0, 2.478750e-05]
    horns: ;; ホーン(稼働部の定義)のリスト
      -
        name: horn
        types: [horn12] ;; 接続可能位置の形名, (A)のリストから選ぶ
        translation: [0, 0, 19]
    fixed-points: ;; 接続可能位置のリスト
      -
        name: left-side-tap
        types: [ bolt12_0-tap ]  ;; 接続可能位置の形名, (A)のリストから選ぶ
        translation: [ 14.25, -16, 0]
        rotation: [0, 1, 0, 90]
;; アクチュエータのない部品の定義 (hornsが無いだけで actuatorsの定義と同じもの)
parts:
```

パーツを増やしたいときは

メッシュを用意して 以下のyamlに記述
config/robot_assembler_parts_settings.yaml ;; パーツ、接続可能点の設定

メッシュファイルは得意な方法で用意する 3D CAD / Blender / EusLispなど

GUI用 (ボタンを増やす)
config/robot_assembler_buttons_layout.yaml ;; ボタンGUIの設定 (viewerとROSで通信)

以下のコミットがパーツを増やしている
https://github.com/agent-system/robot_assembler/commit/11c76fc6070fc9130e1d60f43a0950536a2e082e